Undergraduate Honor Thesis

Singular-Value Decomposition and its Applications

Zecheng Kuang[1]*

1 Department of Mathematics, University of California San Diego, 9450 Gilman Dr, 92092 , La Jolla, CA

Abstract: The singular-value decomposition (SVD) is a factorization of a real or complex matrix. During my study in the linear algebra courses, I found it interesting to decompose a matrix and considered there might have many useful applications of SVD in real life. In this thesis, we will discuss several applications of SVD such as recommendation system, imagine compression, handwritten digits classification and the mathematical theorem behind them.

1. Introduction to Singular-Value Decomposition

Singular-value decomposition (SVD) allows an exact representation of any matrix and it is easy to eliminate the less important data in the matrix to produce a low-dimensional approximation. This is meaningful in such applications as image compression and recommendation system. Moreover, the natural of SVD allows to form a subspace spanned by the columns of the matrix, which is useful in the applications such as digit recognition and destroyed image reconstruction.

Let's first discuss what Singular-value decomposition actually is.

1.1. The Definition of Singular-Value Decomposition

Let M be an $m \times n$ matrix, and let r be the rank of M. Then there exists a matrix factorization called Singular-value Decomposition (SVD)[1] of M with the form $U\Sigma V^T$, where

1. U is an $m \times r$ column-orthonormal matrix; That is, each of its columns is a unit vector and the dot product of any two columns is 0.

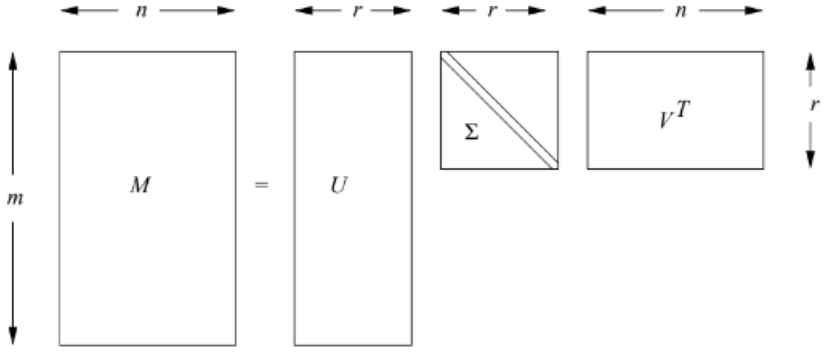
2. Σ is a diagonal matrix where the diagonal entries are called the singular values of M.

3. V^T is an $r \times n$ row-orthonormal matrix; That is, each of its rows is a unit vector and the dot product of any two columns is 0.

* *E-mail: zkuang@ucsd.edu*

[1] *The proof of the existence of SVD can be refered to Applied Numerical Linear Algebra by James W. Demmel [2]*

Figure 1. **Singular-Value Decomposition**



The SVD of a matrix M has strong connections to the eigenvectors of the matrix M^TM and MM^T.

Proposition 1.1.
For any matrix M, M^TM and MM^T have non-negative eigenvalues.

Proof. Suppose $\vec{v}$ is an eigenvector of M^TM whose corresponding eigenvalue is λ, then we have

$$M^TM\vec{v} = \lambda\vec{v}.$$

Multiplying both side by $\vec{v}^T$, we get

$$\vec{v}^TM^TM\vec{v} = \lambda\vec{v}^T\vec{v}.$$

This is equivalent to

$$||M\vec{v}||_2^2 = \lambda||\vec{v}||_2^2.$$

As $||M\vec{v}||_2^2, ||\vec{v}||_2^2 \geq 0$, $\lambda \geq 0$.

Similarly, we can show MM^T has non-negative eigenvalues. □

Proposition 1.2.
For any matrix M, rank(M)=rank(M^TM)=rank(MM^T).

Proof. Let $Null(M)$ be the null space of matrix M and $x \in Null(M)$, we have

$$Mx = 0 \Rightarrow M^TMx = 0.$$

So $x \in Null(M^TM)$, indicating $Null(M) \subset Null(M^TM)$.

Now, suppose $x \in Null(M^T M)$, we then have

$$M^T Mx = 0 \Rightarrow x^T M^T Mx = 0 \Rightarrow ||Mx||_2^2 = 0 \Rightarrow Mx = 0.$$

So $x \in Null(M)$, indicating $Null(M^T M) \subset Null(M)$.

We have $Null(M) = Null(M^T M) \Rightarrow dim(Null(M)) = dim(Null(M^T M)) \Rightarrow rank(M) = rank(M^T M)$.

Since $rank(M) = rank(M^T)$, we also have $rank(M^T) = rank(M^T M)$

Now let's substitute M with M^T, we obtain

$$rank((M^T)^T) = rank((M^T)^T M^T).$$

Equivalently,

$$rank(M) = rank(MM^T).$$

□

Fact 1.1.
The rows of V^T are the eigenvectors corresponding to the positive eigenvalues of $M^T M$ and the squares of the diagonal entries of Σ are the positive eigenvalues of $M^T M$. More specifically, the i^{th} row of V^T is the eigenvector of $M^T M$ whose corresponding eigenvalue is the square of the i^{th} entry of Σ.

Proof. Let's begin our explanation with the transpose of M.

Suppose $M = U\Sigma V^T$, then

$$M^T = (U\Sigma V^T)^T = V\Sigma^T U^T.$$

Since Σ is a diagonal matrix, its transpose is itself. Thus we have

$$M^T = V\Sigma U^T.$$

Multiplying M and M^T, we have

$$M^T M = V\Sigma^2 V^T.$$

Multiplying both sides by V, we get

$$M^T MV = V\Sigma^2.$$

It's not hard to see the diagonal entries of Σ^2 are the eigenvalues of $M^T M$ and V is the matrix whose columns are the corresponding eigenvectors of $M^T M$, which indicates that the rows of V^T are the corresponding eigenvectors of $M^T M$.

Moreover, since $M^T M$ is a symmetric matrix, the number of non-zero eigenvalues equals the rank of the matrix, which is equals to rank of M, which is r. By Proposition 1.1, we know $M^T M$ has r positive eigenvalues. So the diagonal entries of Σ are the square roots of the positive eigenvalues of $M^T M$.

□

Fact 1.2.
The columns of U are the eigenvectors corresponding to nonzero eigenvalues of MM^T.

Proof. Since we have

$$MM^T = (U\Sigma V^T)(V\Sigma U^T) = U\Sigma^2 U^T,$$

multiplying both sides by U, we will get

$$MM^T U = U\Sigma^2 U^T U = U\Sigma^2.$$

We discover that U is the matrix whose columns are the corresponding eigenvectors of MM^T. □

Moreover, there is a relationship between the eigenvalues of $M^T M$ and the eigenvalues of MM^T.

Proposition 1.3.
For any matrix M, $M^T M$ and MM^T have the same nonzero eigenvalues.

Proof. Suppose $(\vec{v}, \lambda)$ is the eigenpair of $M^T M$ and $\lambda \neq 0$,

We then can write

$$M^T M\vec{v} = \lambda\vec{v}.$$

Multiplying both side by M, we have

$$MM^T M\vec{v} = M\lambda\vec{v} = \lambda M\vec{v}.$$

Let $M\vec{v} = \vec{x}$, then

$$MM^T\vec{x} = \lambda\vec{x}.$$

It turns out that λ is also an eigenvalue of MM^T.

Now suppose $(\vec{v}, \lambda)$ is the eigenpair of MM^T and $\lambda \neq 0$,

We can write

$$MM^T\vec{v} = \lambda\vec{v}.$$

Multiplying both side by M^T, we have

$$M^T MM^T\vec{v} = M^T\lambda\vec{v} = \lambda M^T\vec{v}.$$

Let $M^T\vec{v} = \vec{x}$, then

$$M^T M\vec{x} = \lambda\vec{x}.$$

It turns out that λ is also an eigenvalue of $M^T M$.

So we conclude that the nonzero eigenvalues of $M^T M$ and MM^T are the same. □

Remark:

Since M^TM is an $n \times n$ matrix and MM^T is an $m \times m$ matrix, where n and m don't necessarily equal to the r. In fact, n and m are at least as large as r, which indicates that M^TM and MM^T should have an additional $n-r$ and $m-r$ eigenpairs with eigenvalues of zeros. In this paper, we use the reduced SVDs[11], which get rid of the zero eigenvalues and the corresponding eigenvectors, forming U as a $m \times r$ matrix, V^T as $r \times n$ matrix and Σ as $r \times r$ (The diagonal entries of Σ are non-zero singular values of M).

The full SVD of a $m \times n$ matrix is defined as $M = U\Sigma V^T$, where

U is an $m \times m$ matrix, which contains all the eigenvectors of MM^T, including the eigenvectors corresponding to the potential zero eigenvalues.

Σ is an $m \times n$ matrix, whose entry at i^{th} row, i^{th} column is the square root of eigenvalue (possibly zero) of M^TM.

V^T is an $n \times n$ matrix, which contains all the eigenvectors of M^TM, including the eigenvectors corresponding to the potential zero eigenvalues.

Example 1.1.

Let $M = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix}$, the reduced SVD of M is given by

$$\underset{U}{\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}} \underset{\Sigma}{\begin{bmatrix} \sqrt{2} & 0 \\ 0 & \sqrt{2} \end{bmatrix}} \underset{V^T}{\begin{bmatrix} \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} & 0 \\ 0 & \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} \end{bmatrix}}.$$

The full SVD is given by

$$\underset{U}{\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}} \underset{\Sigma}{\begin{bmatrix} \sqrt{2} & 0 & 0 & 0 \\ 0 & \sqrt{2} & 0 & 0 \end{bmatrix}} \underset{V^T}{\begin{bmatrix} \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} & 0 \\ 0 & \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} & 0 \\ 0 & -\frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} \end{bmatrix}}.$$

2. Applications of Singular-Value Decomposition

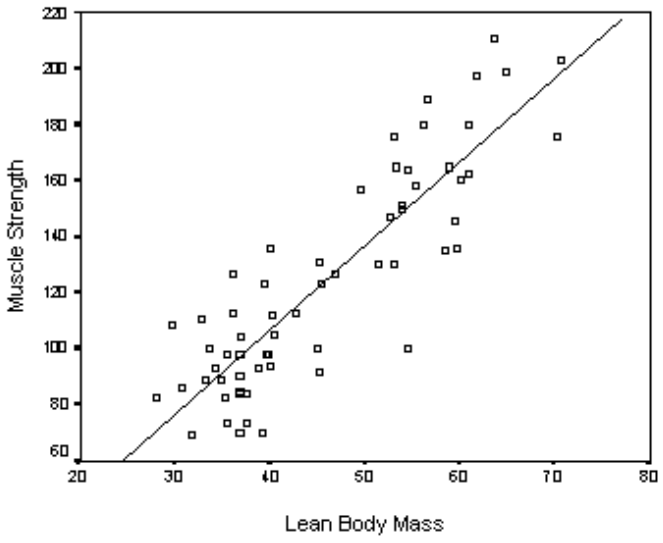
2.1. Least Square Problems

In real life, there might be times when we want to find a "best-fitting" curve to a set of some given points. For example, people might want to find the relationship between lean body mass of human bodies and their muscle strength or analyze how the housing price changes by years to obtain an estimation of the price in the future. Most of the time, we might not be able to find such a linear equation that can be satisfied by all of those points, so finding a "closest" solution is the best we can do. We call these solution to be the least square solutions.

How could we actually find such a "best-fitting curve"?

First let's turn it into a mathematically defined problem.

Figure 2. **Least square solution of the relation between human lean body mass and muscle strength.** [1]



Question 2.1.

Suppose we denote all the points on the graph as (x_i, y_i) pair, where x_i and y_i are the x-coordinate and y-coordinate of point i. We want to find a line $ax + by = c$, such that

$$\sum_i (ax_i + by_i - c)^2$$

is closest to 0, which is the same as finding

$$\min ||a \begin{bmatrix} x_1 \\ x_2 \\ \vdots \end{bmatrix} + b \begin{bmatrix} y_1 \\ y_2 \\ \vdots \end{bmatrix} - c||_2$$

for every $a, b, c \in \mathbb{R}$.

If we let $A = \begin{bmatrix} x_1 & y_1 & -1 \\ x_2 & y_2 & -1 \\ \vdots & \vdots & \vdots \end{bmatrix}$, $\vec{x} = \begin{bmatrix} a & b & c \end{bmatrix}$, the question then becomes to find $\vec{x} \in \mathbb{R}^3$ such that $||A\vec{x}||_2$ is smallest.

In general, we could have a least-square problem as following:

Question 2.2.

Suppose given A and $\vec{b}$, where A is a $m \times n$ matrix and $\vec{b} \in \mathbb{R}^m$, we want to find $\vec{x}$ such that $A\vec{x}$ is closest to $\vec{b}$. In order words, find $\vec{x}$ such that

$$\min_{x \in \mathbb{R}^n} ||A\vec{x} - \vec{b}||_2.$$

$\vec{x}$ is called the least square solution to the equation $A\vec{x} = \vec{b}$.

How could we get the solution?

Firstly, let's introduce the **pseudoinverse** of a matrix.

Definition 2.1 (Pseudoinverse of a Matrix[2]).
Suppose M is an $m \times n$ matrix and the singular-value decomposition of M is given by $U\Sigma V^T$ (either full SVD or reduced SVD), then the **pseudoinverse** of M is given by

$$M^+ = V\Sigma^+ U^T,$$

where Σ^+ is from taking the transpose of Σ and taking the numerical inverse of all the non-zero entries at i^{th} row and i^{th} column for every i.

Example 2.1.

Suppose $\Sigma = \begin{bmatrix} 3 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$, then $\Sigma^+ = \begin{bmatrix} \frac{1}{3} & 0 & 0 \\ 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$.

Theorem 2.1.
Suppose given A and $\vec{b}$, where A is a $m \times n$ matrix and $\vec{b}$ in $\mathbb{R}^m$, then

$$\vec{x} = A^+\vec{b}$$

is the least square solution to $A\vec{x} = \vec{b}$.

Proof. In this proof, we need to utilize the full version of SVD, where U is $m \times m$, V^T is $n \times n$ and Σ is $m \times n$. Let $A = U\Sigma V^T$ be the full SVD of matrix A, since U is a column-orthonormal matrix, then $U^T U = I_m$. As U is a square matrix, U^T has to be the inverse of U, which shows $UU^T = I_m$.

Then for any $\vec{x} \in \mathbb{R}^n$,

$$||A\vec{x} - \vec{b}||_2 = ||U\Sigma V^T\vec{x} - UU^T\vec{b}||_2 = ||U(\Sigma V^T\vec{x} - U^T\vec{b})||_2.$$

Since finding an $\vec{x}$ such that $||U(\Sigma V^T\vec{x} - U^T\vec{b})||_2$ is smallest is the same as finding $\vec{x}$ such that $||\Sigma V^T\vec{x} - U^T\vec{b}||_2$ is smallest because $||U(\Sigma V^T\vec{x} - U^T\vec{b})||_2 \leq ||U||.||\Sigma V^T\vec{x} - U^T\vec{b}||_2$.

For convenience, if we write $V^T\vec{x}$ as $\vec{y}$ and $U^T\vec{b}$ as $\vec{c}$, then if we can find $\vec{y}$ such that $||\Sigma\vec{y} - \vec{c}||$ is smallest, then we can find such $\vec{x}$ accordingly.

Since the columns of Σ are orthogonal, then the columns of Σ form a subspace of $\mathbb{R}^n$, then the only way that can make $||\Sigma\vec{y} - \vec{c}||$ smallest is to make $\Sigma\vec{y}$ the **projection** of $\vec{c}$ onto the column space of Σ.

Definition 2.2 (Projection of a vector onto a subspace).
Let W be a subspace of $\mathbb{R}^n$, $\{u_1, \ldots, u_m\}$ be an orthogonal basis for W. If v is a vector in $\mathbb{R}^n$, the projection of

v onto W is denoted $proj_W v$, where

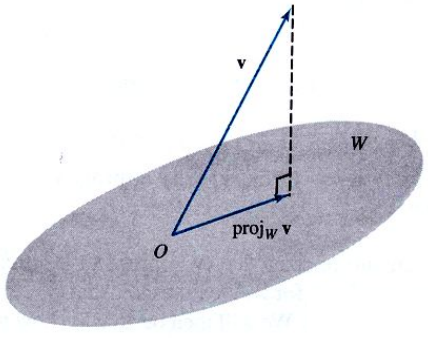
$$proj_W v = \frac{< v, u_1 >}{< u_1, u_1 >} u_1 + \frac{< v, u_2 >}{< u_2, u_2 >} u_2 + \ldots + \frac{< v, u_m >}{< u_m, u_m >} u_m$$

If $\{u_1, \ldots, u_m\}$ is an orthonormal basis for W, then

$$proj_w v = < v, u_1 > u_1 + < v, u_2 > u_2 + \ldots + < v, u_m > u_m$$

.

Figure 3. **Projection of a vector onto a subspace [13]**



Let $C(\Sigma)$ =span $\left\{ \begin{bmatrix} \sigma_1 \\ 0 \\ 0 \\ \vdots \end{bmatrix}, \begin{bmatrix} 0 \\ \sigma_2 \\ 0 \\ \vdots \end{bmatrix}, \ldots, \begin{bmatrix} 0 \\ \vdots \\ \sigma_r \\ \vdots \end{bmatrix} \right\}$, the column space of Σ, where r is the rank of A. And we denote ω_i as the i^{th} column of Σ. So $\Sigma\vec{y}$ must equal to

$$proj_{C(\Sigma)} c = \frac{< c, \omega_1 >}{< \omega_1, \omega_1 >} \omega_1 + \frac{< c, \omega_2 >}{< \omega_2, \omega_2 >} \omega_2 + \ldots + \frac{< c, \omega_r >}{< \omega_r, \omega_r >} \omega_r$$

Since $< \omega_i, \omega_i >= \sigma_i^2$, we can rewrite $proj_{C(\Sigma)} c$ as

$$< c, \frac{\omega_1}{\sigma_1^2} > \omega_1 + < c, \frac{\omega_2}{\sigma_2^2} > \omega_2 + \ldots + < c, \frac{\omega_r}{\sigma_r^2} > \omega_r.$$

Keep in mind that that are n columns in Σ, where there are $n - r$ columns with all zero entries. So it turns out that $< c, \frac{\omega_i}{\sigma_i^2} >= 0$ if $r < i \leq n$.

Now we can write

$$proj_{C(\Sigma)} c = \sum_{i=1}^{n} < c, \frac{\omega_i}{\sigma_i^2} > \omega_i$$

which is the same as the product:

$$\begin{bmatrix} \omega_1 & \omega_2 & \dots & \omega_n \end{bmatrix} \begin{bmatrix} < c, \frac{\omega_1}{\sigma_1^2} > \\ < c, \frac{\omega_2}{\sigma_2^2} > \\ \vdots \\ < c, \frac{\omega_n}{\sigma_n^2} > \end{bmatrix} = \begin{bmatrix} \omega_1 & \omega_2 & \dots & \omega_n \end{bmatrix} \begin{bmatrix} \omega_1^T/\sigma_1^2 \\ \omega_2^T/\sigma_2^2 \\ \vdots \\ \omega_n^T/\sigma_n^2 \end{bmatrix} c.$$

It turns out

$$\Sigma\vec{y} = proj_{C(\Sigma)}c = \Sigma\Sigma^+c.$$

So

$$V^T\vec{x} = \Sigma^+U^T\vec{b}.$$

Multiplying both side by V, we have

$$VV^T\vec{x} = \vec{x} = V\Sigma^+U^T\vec{b}$$

We conclude that

$$\vec{x} = A^+\vec{b}$$

is the least square solution to $A\vec{x} = \vec{b}$.

Remark: In this case, $U\Sigma V^T$ is the full SVD of matrix A and $A^+ = V\Sigma^+U^T$. However, if we let the reduced SVD of A to be $U'\Sigma'V'^T$, then A^+ is also equal to $V'\Sigma'^+U'^T$ because there are $m-r$ all-zero columns and $n-r$ all-zero rows in Σ. When computing the product of $V\Sigma^+$, the last $n-r$ columns of V don't have any effect on the product and thus the product will lead to an $n \times m$ matrix whose last $m-r$ columns are all zeros. When timing the matrix U^T, again the last $m-r$ rows don't have any effect on the result. So the product of $V\Sigma^+U^T$ must be the same as the product of $V'\Sigma'^+U'$.

□

2.2. Image Compression

There are cases where compression of files or images are necessary during transmission between devices. For example, when we browse images on internet with our cellphones, images might be resized to better fit on the screens or compressed to increase efficiency of transmission.

We know an image is composed of pixels and the color of each pixel is composed of three primary colors: Red, Green and Blue. Then any image could be represented as three metrics R,G and B whose entries are the color levels at the corresponding positions.

To better explain the idea, we use the black and white image for our example whose representing matrix is the one with the grey levels at each pixel as entries.

Suppose we have an image of Triton[2].

Figure 4. **Original Image of Triton**

Figure 5. **Black and White Image of Triton**

If we have the matrix M whose entries are the grey values at the corresponding positions of this image. We claim that $M^{'}$ is the k-rank approximation of M, where

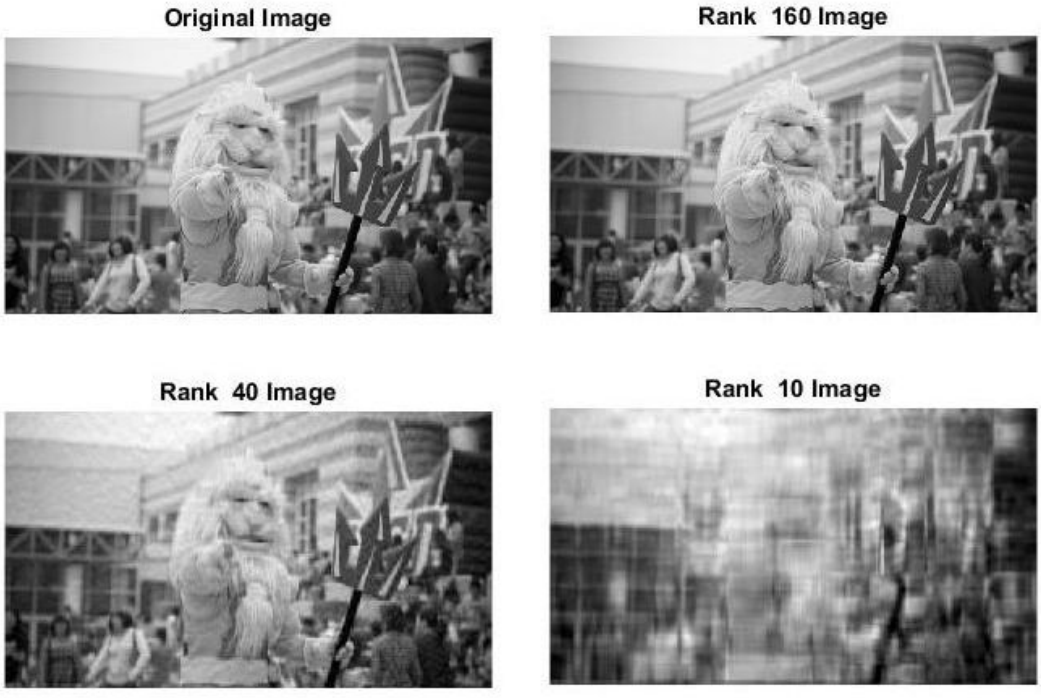
$$M^{'} = U^{'}\Sigma^{'}V^{'T}$$

$U^{'}$ is the k-rank approximation of U, which only has the first k columns of U.
$\Sigma^{'}$ only has the first k singular values of Σ.
$V^{'T}$ only has the first k rows of V^{T}.

If we set a value k and take the best k-rank approximation to the image matrix, we can then obtain an approximate image matrix corresponding the resulting compressed image. Let $k = 160, 40, 10$ respectively, using the matlab code 2.2, we get 3 compressed images as shown in Figure 6.

[2] *Downloaded from http://ucsdnews.ucsd.edu/slideshow/page/welcome_week/2011*

Figure 6. **Compressed Image of Triton**



Since we drop a few singular values, when we recompute the approximated matrix, we only need to deal with the columns of U and rows of V^T that are actually get used. In other words, only first k columns of U, first k singular values and first k rows of V^T need to be stored, and thus we reduce memory usage.

But why would this work?

Proposition 2.1 ([3]).
Let $U\Sigma V^T$ be the SVD of M, then suppose M' is the the product of $U\Sigma' V^T$, where Σ' is constructed by taking the last $r-k$ singular values of Σ to be zero. We say M' is the best k-rank approximation of M, then $||M-M'||_2$ is equal to the sum of the squares of all the singular values that are set to zero.

Proof. Suppose $M = U\Sigma V^T$.

Let $m_{ij}, u_{ij}, s_{ij}, v_{ij}^T$ be the entry in i^{th} row, j^{th} column of matrix M, U, Σ, V^T respectively.

By the definition of matrix multiplication,

$$m_{ij} = \sum_k \sum_l u_{ik} s_{kl} v_{lj}^T$$

Then the square of Frobenius norm of M,

$$||M||^2 = \sum_i \sum_j (m_{ij})^2 = \sum_i \sum_j (\sum_k \sum_l u_{ik} s_{kl} v_{lj}^T)^2 \tag{1}$$

As we square a sum of terms, we can create two copies of the sum and multiply each term of the first sum by each term of the second sum.

$$(\sum_k \sum_l u_{ik} s_{kl} v_{lj}^T)^2 = (\sum_k \sum_l u_{ik} s_{kl} v_{lj}^T)(\sum_m \sum_n u_{im} s_{mn} v_{nj}^T) = \sum_k \sum_l \sum_m \sum_n u_{ik} s_{kl} v_{lj}^T u_{im} s_{mn} v_{nj}^T$$

So equation(1) can be written as

$$||M||^2 = \sum_i \sum_j \sum_k \sum_l \sum_m \sum_n u_{ik} s_{kl} v_{lj}^T u_{im} s_{mn} v_{nj}^T \tag{2}$$

Since Σ is a diagonal matrix, that is s_{kl} and $s_{mn} = 0$ unless $k \neq l$, $m \neq n$.

So we can rewrite equation(2) as

$$||M||^2 = \sum_i \sum_j \sum_k \sum_m u_{ik} s_{kk} v_{kj}^T u_{im} s_{mm} v_{mj}^T$$

Since U is column-orthonormal, $\sum_i u_{ik} u_{im} = 1$ if $k = m$ or 0 otherwise. Then

$$||M||^2 = \sum_j \sum_k s_{kk} v_{kj}^T s_{kk} v_{kj}^T$$

Since V^T is row-orthonormal, $\sum_j v_{kj}^T v_{kj}^T = 1$. Therefore,

$$||M||^2 = \sum_k (s_{kk})^2 \tag{3}$$

Suppose we have $M' = U'\Sigma' V^{T'}$, which preserves the first d singular values of Σ, then since $U'\Sigma' V^{T'} = U\Sigma' V^T$, $M - M' = U(\Sigma - \Sigma')V$.

By equation(3), we have the Frobenius norm of M-M'

$$||M - M'||^2 = \sum_k (s_{kk} - s'_{kk})^2 \tag{4}$$

where s'_{ij} is the entry in i^{th} row and j^{th} column of Σ'.

Since the only k such that s_{kk} differ to s'_{kk} are those entries set to be zeros in Σ'. That is, $||M - M'||^2$ is the sum of the squares of the elements of Σ that were set to 0.

□

So to minimize $||M - M'||^2$, we have to pick the smallest elements in Σ to be zero.

In practice, people want to retain 90% of information in the approximation. More specially, we want the sum of the squares of the singular values in Σ' to be at least 90% of the sum of the squares of the singular values in Σ.

Matlab code 2.2 to compress an image with 7 different k-rank approximation (320, 160, 80, 40, 20, 10, 5) :

```
function [ output_args ] = CompressImage( filename )
    %open an image
    image = rgb2gray(imread(filename));
    image = im2double(imresize(image,0.5));

    %display the original image
    subplot(4, 2, 1), imshow(image), title(sprintf('Original Image'));

    %get the SVD of the image
    [U, S, V] = svd(image);

    %get the diagonal entries of sigma
    diagonals = diag(S);

    ranks = [320,160,80,40,20,10,5]
    for i=1:length(ranks)
        %get the new sigma
        compressed_sigmas = diagonals;
        compressed_sigmas(ranks(i):end) = 0;

        compressed_S = S;
        n = length(diagonals);
        compressed_S(1:n, 1:n) = diag(compressed_sigmas);

        %calculate the compressed image
        approx_image = U * compressed_S * V';

        %display the compressed image
        subplot(4, 2, i+1), imshow(approx_image), title(sprintf('Rank %d Image',
            ranks(i)));
    end
end
```

2.3. Recommendation System [3]

Companies such as Amazon and Netflix collect tons of data, for example, users' browsing or purchasing records and provide recommendation to them according to some analyses on the data. How to know the user better and recommend them the products they truly like then becomes a crucial problem that needs an effective solution. In this chapter, we will discuss the application of singular-value decomposition in recommendation system and explain how it works.

Suppose we are given the ratings of seven people towards five different movies respectively,

User|Movie	Matrix	Inception	Stars Wars	Casablanca	Titanic
John	1	1	1	0	0
Billy	3	3	3	0	0
Charlie	4	4	4	0	0
Bella	5	5	5	0	0
Jack	0	0	0	4	4
Alex	0	0	0	5	5
Harry	0	0	0	2	2

We can then represent these ratings as a matrix M whose columns are the ratings of each user respectively. If we compute the SVD of this matrix M, we will get

$$M = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 3 & 3 & 3 & 0 & 0 \\ 4 & 4 & 4 & 0 & 0 \\ 5 & 5 & 5 & 0 & 0 \\ 0 & 0 & 0 & 4 & 4 \\ 0 & 0 & 0 & 5 & 5 \\ 0 & 0 & 0 & 2 & 2 \end{bmatrix}$$

$$= \underbrace{\begin{bmatrix} 0.14 & 0 \\ 0.42 & 0 \\ 0.56 & 0 \\ 0.70 & 0 \\ 0 & 0.60 \\ 0 & 0.75 \\ 0 & 0.30 \end{bmatrix}}_{U} \underbrace{\begin{bmatrix} 12.4 & 0 \\ 0 & 9.5 \end{bmatrix}}_{\Sigma} \underbrace{\begin{bmatrix} 0.58 & 0.58 & 0.58 & 0 & 0 \\ 0 & 0 & 0 & 0.71 & 0.71 \end{bmatrix}}_{V^T}$$

There is a way to interpret U, Σ, V^T respectively, where

$$= \begin{bmatrix} 0.14 & 0 \\ 0.42 & 0 \\ 0.56 & 0 \\ 0.70 & 0 \\ 0 & 0.60 \\ 0 & 0.75 \\ 0 & 0.30 \end{bmatrix}$$
$$U$$

U: Connects users to movie genres.

Each row represents the how each user likes a movie genre. For example, the numbers 0.70 and 0 in the fourth rows of U represent Bella prefers genre 1 better than genre 2, where genre 1 represents "Sci-fi" and genre 2 represents "Romance" in this case.

$$\begin{bmatrix} 12.4 & 0 \\ 0 & 9.5 \end{bmatrix}$$
$$\Sigma$$

Σ: Each diagonal entry represents the strength of each genre. The strength of "Sci-fi" is greater than the strength of "Romance" because the data provides more information about "Sci-fi" genre and users who like "Sci-fi."

$$\begin{bmatrix} 0.58 & 0.58 & 0.58 & 0 & 0 \\ 0 & 0 & 0 & 0.71 & 0.71 \end{bmatrix}$$
$$V^T$$

V^T: Connects genre to movies.

Each row represents the strength of genre each movie partakes. As we seen, Matrix, Inception and Stars Wars all have a positive numbers of 0.58 as "Sci-fi" genre while Casablanca and Titanic have 0's as they don't partake any concept of "Sci-fi".

Suppose there is new user Jane who only watches Stars Wars and rates it 4. How could we use this system to recommend movies to her?

First of all, We can represent her rating as a vector $u = [0, 0, 4, 0, 0]$. By computing uV, we map Jane's ratings into the "genre space". Since $uV = [2.32, 0]$, Jane is interested in Sci-fi movies, but not interested in "Romance" at all. We now have a representation of Jane's ratings in "genre space".

One useful thing we can do now is to map her representation back into "movie space" by computing

$[2.32, 0]V^T$, because matrix V^T interprets the genre each movie partakes. If we compute $[2.32, 0].V^T$ to get $[1.35, 1.35, 1.35, 0, 0]$, this indicates that Jane would like Matrix, Inception and Star Wars, but not Casablanca or Titanic.

Another approach is to find users who are similar to Jane and provide recommendations to Jane according to the preferences of those similar users. We can use V to map all the users into "genre space". For example, John's ratings will map to $[1.74, 0]$, while Jack's will map to $[0, 5.68]$. We can then measure the similarities between users by their cosine distances in "concept space". It is clear that Jane is more similar to John than to Jack because the $[2.32, 0]$ and $[1.74, 0]$ have the same direction, which gives the cosine distance 1, while $[2.32, 0]$ and $[0, 5.68]$ have a dot product of 0, which gives the cosine distance 0, indicating the angle between them is 90 degree[3]. By looking up John's preferences, we could then provide recommendations to Jane accordingly.

Remark:

In the example explained above, all the users rate either "Sci-fi" or "Romance", but not both. In this case, the rank of matrix M is equal to the number of "genres", which is 2 and the decomposition of M will give exactly the desired number of columns of U and V. However, in real life, we might not have such a simple case. In fact, we might have the rank of M greater than the number of columns we want for U,Σ, V. In this case, we then need to eliminate the smallest singular values and the corresponding columns of U and V to get the approximation of M as long as the approximation retains "enough" information from the original one. [2.1]

2.4. Classification of Handwritten Digits[4]

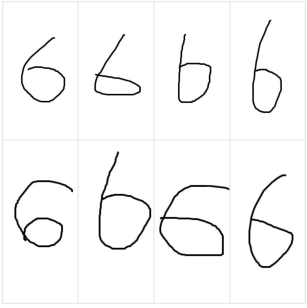
Handwritten digits are very common in our lives and there are cases that people want to recognize the handwritten digits or letters by computers. For example, postal office requires an automated and precise way to recognition handwritten zip codes. This is now automated by Optical Character Recognition (OCR) [10].

In this chapter, we will discuss the application of SVD in handwritten digits classification.

Suppose we have a set of k training images which have been classified as digit i, we want to use these images to form our "recognition system".

[3] *Cosine distance between two vectors u and v is given by* $\cos\theta = \frac{<u,v>}{||u||_2||v||_2}$.

Figure 7. **8 training images of digit 6**



Firstly, we have to cut the frames and resize these k images to get images with size 16x16. Then we construct k matrices whose entries are the grey levels of pixels at corresponding locations of each images respectively.For each of these k matrices, we vectorize them into $\mathbb{R}^{256}$ vectors and use these vectors as columns to construct matrix M. M should be 256 by k. [8][9]

Figure 8. **Vectorization[12]**

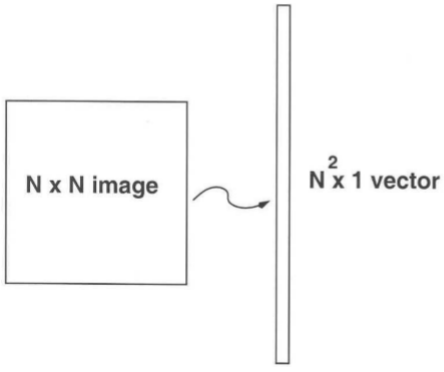
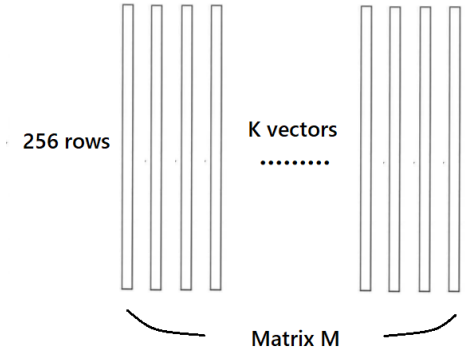


Figure 9. **Construction of matrix M**



Proposition 2.2.
Suppose $M = U\Sigma V^T$, *then*

$$\{u_1, u_2, \ldots, u_r\}$$

form an orthonormal basis of the column space of M, where r is the rank of M and u_i *is the* i^{th} *column of U.*

Proof. Suppose M has rank r, we can write

$$M = U\Sigma V^T = \sigma_1 u_1 v_1^T + \sigma_2 u_2 v_2^T + \ldots + \sigma_r u_r v_r^T$$

where σ_i, u_i, v_i^T are the i^{th} singular value of Σ, i^{th} column of U and i^{th} rows of V^T.
Since each of $\sigma_i u_i v_i^T$ can be written as

$$\sigma_i u_i v_i^T = \sigma_i \begin{bmatrix} \\ u_i \\ \\ \end{bmatrix} \begin{bmatrix} v_i^T{}_1 & v_i^T{}_2 & \ldots & v_i^T{}_n \end{bmatrix} = \begin{bmatrix} \sigma_i v_i^T{}_1 u_i & \sigma_i v_i^T{}_2 u_i & \ldots & \sigma_i v_i^T{}_n u_i \end{bmatrix}$$

where $v_i^T{}_j$ denotes the j^{th} entry of vector v_i^T and n is the number of elements in vector v_i^T. Then we have

$$M = \begin{bmatrix} \sigma_1 v_1^T{}_1 u_1 + \sigma_2 v_2^T{}_1 u_2 + \ldots + \sigma_r v_r^T{}_1 u_r & \ldots\ldots & \sigma_1 v_1^T{}_n u_1 + \sigma_2 v_2^T{}_n u_2 + \ldots + \sigma_r v_r^T{}_n u_r \end{bmatrix}$$

In general the i^{th} column of M can be written as

$$\sigma_1 v_1^T{}_1 u_1 + \sigma_2 v_2^T{}_1 u_2 + \ldots + \sigma_r v_r^T{}_1 u_r$$

and thus each column vector of M can be written as a linear combination of $\{u_1, u_2, \ldots, u_r\}$ □

If we compute the SVD of this matrix M, then the columns of U form an orthogonal basis of the column space of matrix M. In other words, $u_1, u_2, ..., u_r$ form a "digit space" for a specific digit.

Keep in mind that we now only compute the subspace of one digit. We have to continue the same process to compute the subspace of all the ten digits.

Now, if given any unknown image of a digit, we do the pre-processing on this image as explained before and get a vector $\vec{q}$. We want to find the "digit space" that $\vec{q}$ is closest to. The way to do that is to find the smallest **residual** between $\vec{q}$ and all the orthonormal space.
Since $u_1, u_2, ..., u_r$ is an orthonormal basis for the column space of matrix M, denoting this column space W, we define **residual** between $\vec{q}$ and the orthonormal basis as the distance between $\vec{q}$ and $proj_w q$ [2.2], which is given as

$$||\vec{q} - \sum_{i=1}^{r} < q, u_i > u_i||_2$$

Since there are 10 digits in total, we want to compute the residual between $\vec{q}$ and each of these 10 orthonormal basis and classify the unknown digit as d, where the residual between $\vec{q}$ and the orthonormal basis for digit d is smallest among all the ten digits.

In other words, we want to find d, such that

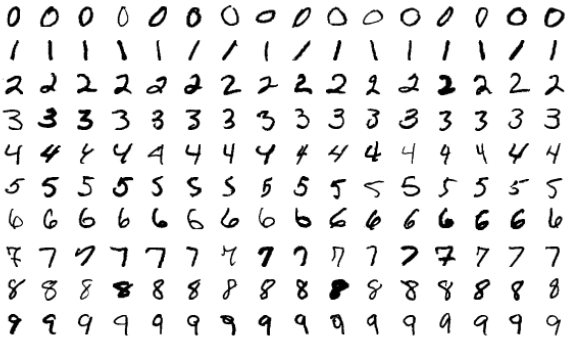
$$\min_{0\leq d\leq 9} ||\vec{q} - \sum_{i=1}^{r_d} < v, u_{d,i} > u_{d,i}||_2$$

And we conclude d is the classification of the unknown digit.

Handwritten digit classification is actually a classic problem in machine learning. There are many different approaches to solving this problem such as principal component analysis(PCA), nearest neighbor method, statistical modelling and neural networks. It is always meaningful to write codes of the algorithm we use for the problem as we can analyze the efficiency and accuracy of such algorithm. We try to use the idea of singular-value decomposition as discussed before to write a Matlab code to classify handwritten digits.

But first we need some pre-classfied training images. Thanks for the MNIST database, we could get a huge set of training images and their classifications. MNIST data database, which is available at [8] contains 60,000 training examples with corresponding classifying labels and 10,000 examples for testing.

Figure 10. **Sample images from MNIST test dataset**[4]



The Matlab code is given below:

The "digitRecognition" function takes two input: the number of training images using for each digit and the number of testing samples we want to classify. The function outputs the percentage of correct classification.

Matlab code for digitRecognition: :

```
function [rate] = digitRecognition( numTrain, numTest )
    d = load('mnist.mat');
```

```
    X = d.trainX;
    Y = d.trainY;
    A = d.testX;
    B = d.testY;

digitSpace=zeros([256,numTrain,10]);

position = ones([1,10]);

i = 0;
while i < (numTrain*10)
    pos = randi(60000);
    digit = Y(1,pos);
    image = reshape(X(pos,:),28,28)';
    imageVec = ImageCrop2(image);
    if position(1,digit+1) <= numTrain
        digitSpace(:,position(1,digit+1),digit+1)=imageVec;
        position(1,digit+1)= position(1,digit+1)+1;
        i = i+1;
    end
end

err = 0;
for i = 1:numTest
    pos = randi(10000);
    unknown = reshape(A(pos,:),28,28)';
    vv = double(ImageCrop2(unknown));

    res = zeros([1,10]);
    for digit = 0:9
        [U,S,V]=svd(double(digitSpace(:,:,digit+1)),'econ');
        rank=length(diag(S));
        res(1,digit+1) = residual(U,rank,vv);
    end

```

```
38      value = find(res==min(res));
39          value = value-1;
40
41      if(value~=B(1,pos))
42          err = err+1;
43      end
44
45  end
46
47  rate = double((numTest-err)/numTest);
48
49  end
```

Helper functions include “ImageCrop” which takes an image as input in the form as a matrix and returns the pre-processing image(cutting the frame) as a vector.

Matlab code for ImageCrop

```
1   function [ vector ] = ImageCrop2( matrix )
2
3       [row col] = size(matrix);
4
5       sumCol = sum(matrix,1);
6       Left = 0;
7       for i = 1:col
8           if sumCol(i)>0
9               Left = i;
10              break;
11          end
12      end
13
14      sumRow = sum(matrix,2);
15      Up = 0;
16      for i = 1:row
17          if sumRow(i)>0
18              Up = i;
19              break;
```

```
        end
    end

    sumCol = sum(matrix,1);
    Right = 0;
    for i = col:-1:1
        if sumCol(i)>0
            Right = i;
            break;
        end
    end

    sumRow = sum(matrix,2);
    Bottom = 0;
    for i = row:-1:1
        if sumRow(i)>0
            Bottom = i;
            break;
        end
    end

    I = matrix;
    I2 = imcrop(I, [ Left Up Right-Left Bottom-Up]);

    I3 = imresize(I2,[16 16]);
    vector = I3(:);

end
```

"residual" function basically returns the residual between a vector and a subspace.

Matlab code for residual

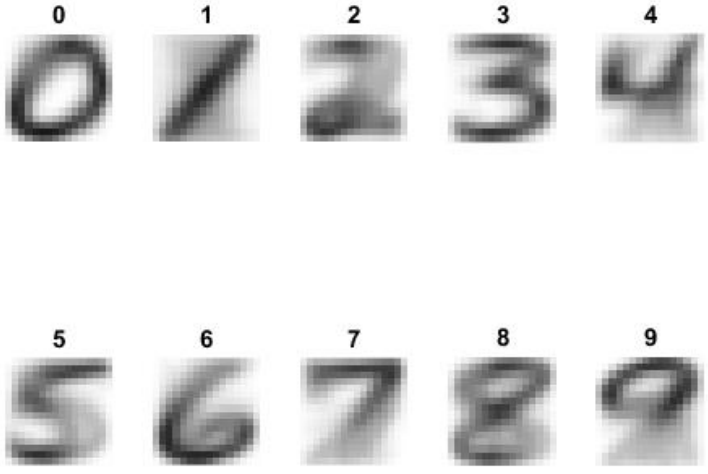
```
function [ res ] = residual(basis, rank, vector )
    proj = 0;
    for i = 1:rank
        proj = proj + (dot(vector,basis(:,i)))*basis(:,i);
```

```
    end
    res = norm(vector-proj);

end
```

When we use 100 training images for each of the 10 digits to form our "digit space", we get the following "mean digits".

Figure 11. **10 Mean Digits when using 100 training images**



These averaged images are called centroids[9]. We are treating each image as a $\mathbb{R}^{256}$ vector, which is vectorized from the matrix of dimension 16×16, and then taking the average of all images in each digit classification individually.

But how many training images do we actually need in practice?
If we fix the number of testing samples to be 1000 and use $(5, 10, 15, 20, \ldots, 100)$ training images for each digit, we can get the corresponding classification percentage.

Figure 12.

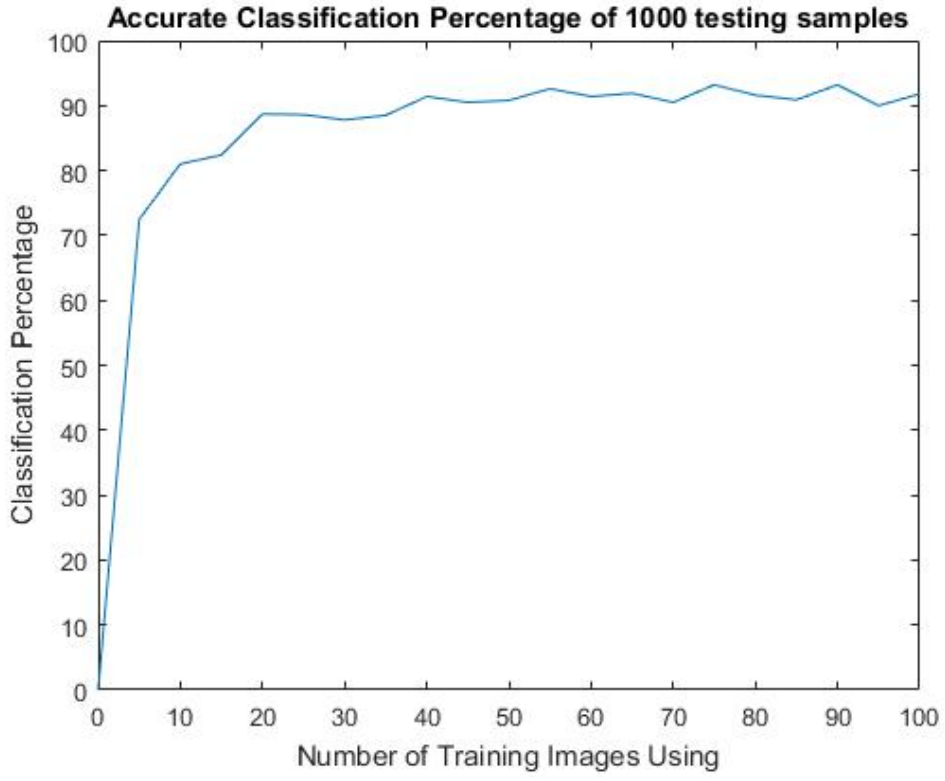
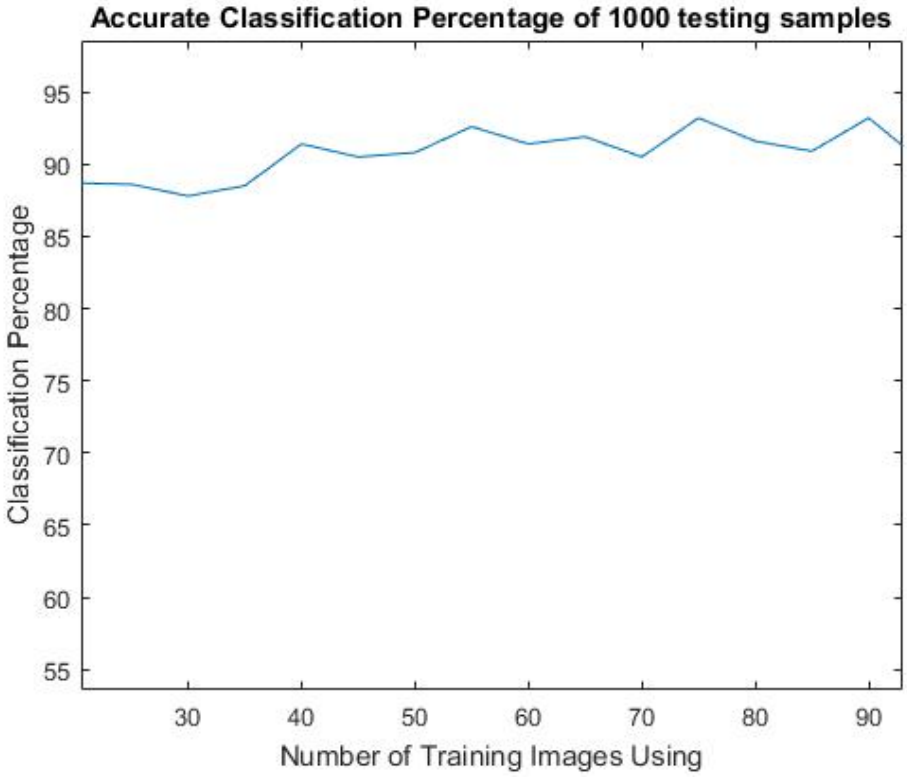


Figure 13. shows the accurate classification rate as a function of number of training images using for each digit. The percentage approaches 90% with 20 training digits and reaches an stable percentage of about 92% after 40 training digits.

Figure 13.



Only about 2% of accuracy of classifying 1000 handwritten digits increases with these extra 20 training digits, which gives about 20 more correct classifications. However, the time used to compute "digit space" using 40 training digits is much longer than that of using 20 training digits. So it might not be necessary to use more than 40 training digits for each digit depending on the accuracy required.

2.5. Reconstruction of Destroyed Images of Human Faces

It is common that accuracy loss happens during files or images transmission between devices. Most people have experienced the situation that the file or image downloaded is destroyed and can't be assessed. Though this is rare in our lives, what could we actual do if we couldn't get assessed to the original file or even the original file doesn't exist anymore? Could we obtain a best approximation of the original file? We adapt the idea from [4][5]

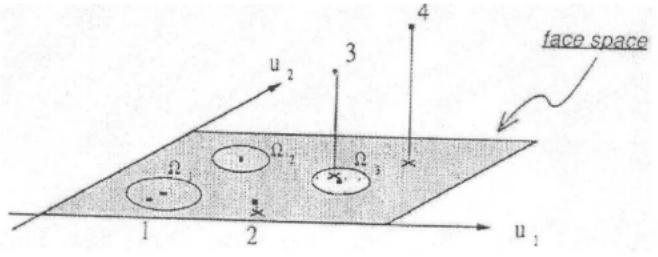
Suppose we have a destroyed image of human face as followed. [5].

[5] *Image from https://www.math.vt.edu/ugresearch/Lassiter_2012_2013.pdf*

Figure 14. **Destroyed Image of Human Face**

Using the similar idea as the "Handwritten Digit Recognition" example, we could construct a "human face space" using different training images of the person's faces (centered and of the same sizes). Viewing each image of a person as one vector and constructing a matrix M using these vectors as columns, we could obtain the "face space", which is the column space of matrix M or more specifically, the space spanned by $\{u_1, u_2, \ldots, u_r\}$, where u_i is the i^{th} column of matrix U given by the SVD of M.

Figure 15. **Face Space** [5]



However, unlike images of digits which are easy distinguishable 2-dimensional pictures, images of human faces are much more complicated in terms of the variations in their facial expressions, the angles of faces towards camera, the different shadows casts caused by different light source direction (known as illumination [6]) and the background of the photo. An individual's identity is captured by all these variations which will cause variances between different images of the same individual. So it is necessary to `normalize` a face in order to minimize the variance [7]. One way to do that is to subtract each image by the "Mean human face".

More specifically, if given $\{v_1, v_2, \ldots, v_k\}$, k human face vectors, we define Mean face vector as:

$$Mean = \frac{1}{k}\sum_{i=1}^{k} v_i.$$

And for each vi, we need to subtract the Mean face

$$\gamma_i = v_i - Mean$$

to get the normalized face vector γ_i.

Then we construct a matrix M whose columns are all the γ_i's. If we compute the SVD of M and get $U\Sigma V^T$, then by Proposition 2.2, we get a space spanned by the column vectors of M and call this "face space".

Now, since we have a destroyed image of human face, if we vectorized it and subtract from Mean, we get a vector, called γ_{des}.

Then the reconstruction is done by getting the best human face approximation to this image, which is the projection of γ_{des} onto the "face space", or

$$reconstruction = projection_{Span\{\gamma_1,\gamma_2,...,\gamma_k\}}\gamma_{des}.$$

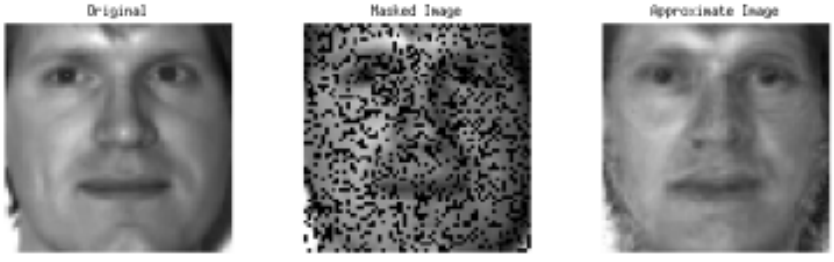
We now re-frame the vector of the reconstruction into a square matrix. This matrix then will represent the recovered image of the human face.

However, we have to normalized all the vectors of faces which means for every v_i, we calculate $v_i - \frac{1}{k}\sum_{n=1}^{k} v_i$.

Then the projection of this destroyed image-Mean onto the face space is the best approximation of this destroyed image to human face image. So we get the recovered image by taking the projection.

The recovered image is given by [4] as below

Figure 16. **Recovered Image of Human Face**



Acknowledgements

First of all, I would like to thank my adviser Thang Huynh for his patient guidance and encouragement during the two quarters of my studying and researching. I am deeply thankful to him for giving me this opportunity and

being my adviser.

I would also like to thank all the professors that I have taken courses with in the Math Department at UCSD. Without their advice and enthusiasm for mathematics, I wouldnt have the passion for mathematics and finish this paper. Especially thanks to Professor Thang Huynh for his advice in the Honors Program and in my pursuit of the beauty of mathematics.

References

[1] G. Dallal, Introduction to Simple Linear Regression, available at http://www.jerrydallal.com/lhsp/slr.htm.

[2] JW. Demmel, Applied Numerical Linear Algebra, Soc. for Industrial and Applied Math, 1997, 109-117.

[3] J. Leskovec, A. Rajaraman, J. Ullman, Mining of Massive Datasets, Cambridge University Press, 2011, 418-427.

[4] M. Mazack. Algorithms for Handwritten Digit Recognition. Masters colloquium, Mathematics Department, Western Washington University, 2009.

[5] M. Turk and A. Pentland, Eigenfaces for Recognition, Journal of Cognitive Neuroscience, vol. 3, no. 1, pp. 71-86, 1991.

[6] T. Chen, W. Yin, X.-S. Zhou, D. Comaniciu, T. S. Huang, Illumination Normalization for Face Recognition and Uneven Background Correction Using Total Variation Based Image Models CVPR, 2005.

[7] T. Jebara, 3D Pose Estimation and Normalization for Face Recognition, Center for Intelligent Machines, McGill University, 1996, 61-73.

[8] Y. LeCun, C. Cortes, C.Burges, The Mnist Database, http://yann.lecun.com/exdb/mnist/.

[9] Wikipedia., Centroids, https://en.wikipedia.org/wiki/Centroid.

[10] Wikipedia., Optical character recognition, https://en.wikipedia.org/wiki/Optical_character_recognition.

[11] Wikipedia., Reduced SVDs, https://en.wikipedia.org/wiki/Singularvalue_decomposition#Reduced_SVDs.

[12] Image downloaded from https://www.vision.jhu.edu/teaching/vision08/Handouts/case_study_pca1.pdf.

[13] Image downloaded from http://slideplayer.com/slide/7531421/.